Um Sistema P2P para Monitorização Eficiente e Automatizada de Redes de Comunicação

An Efficient and Automated P2P Overlay System for Network Monitoring

Ricardo Mendonça
Department of Informatics
University of Minho
Braga, Portugal
pg35393@alunos.uminho.pt

Pedro Sousa
Centro Algoritmi, Department of Informatics
University of Minho
Braga, Portugal
pns@di.uminho.pt

Resumo — **Uma monitorização precisa e eficiente é vital para garantir que uma rede funcione de acordo com o comportamento pretendido, tal como atuar rapidamente face aos problemas encontrados. A tarefa de monitorização de uma rede torna-se complexa com o aumento do tamanho e heterogeneidade da infraestrutura. Este trabalho tem como objetivo aprofundar o desenvolvimento de um sistema *Peer-to-Peer* (P2P) para monitorização de redes. Serão descritos todos os processos de criação do sistema e especificadas as entidades existentes e seus processos de comunicação. De seguida, expor-se-á o processo de definição de uma metalinguagem com o objetivo de possibilitar aos administradores a configuração e pré-programação da rede de monitorização de forma eficaz e versátil. Será dado também relevo ao processo de otimização do sistema, com o objetivo de reduzir o tráfego gerado pela rede P2P. Adicionalmente, será abordado como o sistema é tolerante a falhas, recuperando o estado caso alguma entidade do sistema tenha algum problema.**

Palavras Chave - P2P, Redes Overlay, Deteção de anomalias de rede, ISP, Monitorização de redes.

Abstract — **Precise and efficient monitoring is vital to ensure that a network works according to the intended behaviour, as well as quickly acting in response to possible problems. The task of monitoring a network becomes complex with increasing network size and its heterogeneity. This work explores the development of a *Peer-to-Peer* (P2P) system to monitor network communication infrastructures. All the processes of creation of the system and the existing entities will be described as well as their communication capabilities. Then, the definition of a metalanguage will be exposed, allowing network administrators to configure and pre-program the network in an effective and versatile manner. The optimization of the monitoring system will be also described, reducing the traffic generated by the P2P network. It will be also addressed how the system became fault-tolerant, recovering its state even if any entity has a problem.**

Keywords - P2P, Overlay Networks, Network Anomaly Detection, ISP, Network monitoring.

I. INTRODUÇÃO

As redes de comunicação deparam-se algumas vezes com anomalias que afetam o serviço oferecido aos utilizadores finais. Algumas dessas anomalias relacionam-se com falhas temporárias ou permanentes de equipamento físico da rede (e.g. links, routers, etc.) que implicam uma degradação do nível de serviço oferecido. Outras anomalias resultam de situações de congestão originada por níveis excessivos de tráfego em alguns pontos críticos da rede. A prática atual de monitorização de rede depende, em grande parte, de operações manuais e, assim, as empresas gastam uma parte significativa dos seus orçamentos na parte que monitoriza as suas redes [1]. As soluções de monitorização e gestão de rede disponíveis não são apenas caras, mas também difíceis de usar, configurar e manter [2]. Este trabalho visa a especificação e desenvolvimento de um sistema *Peer-to-Peer* (P2P) que possibilite a deteção de anomalias de rede, podendo ser usado pelos administradores dos *Internet Service Providers* (ISPs). Existem várias alternativas distintas para criar uma rede *overlay* P2P, sendo o sistema proposto centralizado [3]. Existem outros projetos relacionados, que também retiram vantagens de uma infraestrutura de *probings* distribuída, como os projetos: *Ripe Atlas* [9], o *perfSonar* [10], o *Sam Knows* [11], e o *NLNOG Ring* [12].

O sistema proposto é composto por um conjunto de *peers* distribuídos na rede e um nó coordenador responsável pela gestão da rede P2P, pela coleta de informação de anomalias ocorridas na rede do ISP, e por disponibilizar uma interface com o administrador. Este poderá iniciar a monitorização em *links* de uma rede com alguns tipos de medições, podendo alterar vários parâmetros. A rede P2P disponibiliza alarmes ao administrador para situações tais como: falta de conectividade em determinadas *paths edge-to-edge*; situações de alteração de rotas; degradação crítica do atraso, perdas de pacotes, congestão, etc. Serão explorados com especial foco os seguintes aspetos relativos à operação da rede de monitorização: *i)* redução do tráfego gerado pela rede P2P através da composição dos resultados obtidos por diferentes *peers* (e.g. possibilidade de agregação/composição dos resultados obtidos por diferentes *peers* para obter informação relativa a uma determinada *path*); *ii)* estratégias de programação da rede P2P para tomar determinadas ações

autónomas e possibilitar a adaptação a diferentes cenários de operação; *iii)* procedimentos para tornar o sistema tolerante a falhas ao nível da rede ou da própria *overlay* de monitorização e *iv)* análise de resultados ilustrativos dos mecanismos.

Este documento é composto por 5 secções que estão organizados da seguinte forma: na secção I foi feito o enquadramento e contextualização do trabalho. Na secção II é detalhada a arquitetura do sistema. A secção III descreve os passos da criação de vários mecanismos desenvolvidos para o sistema. Na secção IV é utilizado um emulador de redes para testar os mecanismos desenvolvidos. Por último, na secção V, serão apresentadas as principais conclusões e apresentadas algumas possíveis melhorias do sistema para trabalho futuro.

II. Arquitetura, Entidades e Comandos de *Probing*

Nesta secção é apresentada a arquitetura e entidades presentes no sistema e exemplos de comandos que administrador tem disponíveis para comunicar com sistema. O coordenador, tem a função de coordenar todos os *peers* através de vários comandos, podendo iniciar *probings* com diversos parâmetros opcionais. Esta entidade, também permite verificar os resultados dos *probings*, que podem ser posteriormente analisados numa interface gráfica, onde se poderá verificar se algum dos alarmes definidos pelo administrador foi despoletado. A entidade *peer*, está encarregue de recolher informações sobre a rede e enviá-las ao coordenador. Num *probing*, existe um *peer* origem, que recebe os comandos do coordenador e envia os dados recolhidos da *path* entre esse *peer* origem e um *peer* destino, definidos ao iniciar um *probing* no coordenador. Na Figura 1, dois *probings* podem ser observados entre o *peer* A e B, e entre o *peer* A e C, onde os resultados estão a ser enviados a uma porta de transferência de dados do coordenador para posterior análise.

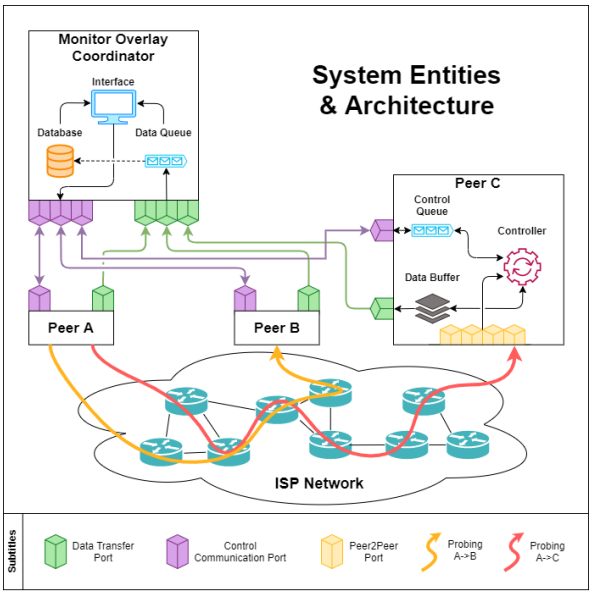

Figura 1. Arquitetura e entidades do sistema P2P de monitorização

O coordenador da rede tem várias portas de comunicação de controlo, que permitem enviar diversos comandos para os *peers* existentes na rede, este, também tem várias portas de transferência de dados que permite receber as várias medições que os *peers* estão a coletar, sendo estas medições guardadas numa base de dados de grafos implementada em *Neo4j*. Esta entidade tem uma interface para os administradores enviarem comandos, receber notificações de alarmes e verificar o estado da rede do ISP, tal como está ilustrado na Figura 1.

A entidade *peer*, tem dois canais distintos, um para controlo que é preciso para receber pacotes *Transmission Control Protocol* (TCP) com comandos que vão correr no *peer*, e enviar pacotes de resposta de confirmação de volta ao coordenador, onde será necessária uma *queue* de controlo que funcionará nas duas direções. O outro canal é usado para transferência de dados através da comunicação constante com pacotes TCP até ao coordenador, para monitorização contínua da rede. Um *buffer* de dados também irá ser necessário, para temporariamente guardar os dados enquanto se transfere a informação do *peer* ao coordenador. Uma lista de *peers* é necessária para permitir *probing* entre vários *peers*. Os *peers* têm um controlador que será preciso para cada *peer* saber que medições são necessárias ser feitas (indicadas previamente pelo coordenador), como ilustrado na Figura 1. É possível enviar vários comandos do coordenador para os *peers*. Estes comandos têm vários parâmetros opcionais para permitir ao administrador escolher a melhor maneira de monitorizar a rede e definir os alarmes de diversas formas. Apresentam-se de seguida só alguns exemplos de comandos definidos:

- **Round Trip Time (RTT)** *Number of Packets, Size, ICMP, Timeout, Loop, Feedback(Feedback Time) ou Statistic Options, Peer Y, Alarm(Alarm Condition)*: Permite a medição do round trip time entre dois peers diferentes (*peerX -> peerY*). A *flag Internet Control Message Protocol* (ICMP) está ativa por defeito (1), mas o administrador pode desativá-la mudando o seu valor para 0. Quando ativa, esta medição vai usar um *echo-request* para medir o RTT. Se a *flag* ICMP está inativa (0), o *peer* vai usar uma implementação de round trip time puramente criada em Java usando o protocolo *User Datagram Protocol* (UDP);
  Outros comandos afins lidando com outras métricas (perdas, *jitter*, etc. [4]) são também suportados.
- **RoutePath** *Timeout, Loop, Feedback(Feedback Time) ou Statistic Options, PeerY*: Este comando permite ao coordenador obter a *route path* do *peerX* ao *peerY*. Uma chamada ao sistema é usada para medir este processo. Isto faz com que seja possível verificar quando ocorrem mudanças em especificas *route paths* do ISP, e quanto tempo essas mudanças persistem, o que pode indicar uma possível anomalia;
- **Cancel** *Probe ID, Timeout, PeerY*: Permite ao administrador cancelar indefinidamente um processo de medição que está a decorrer num *peer*. Depois de receber este comando e quando a informação está pronta a enviar, o *peer* vai enviá-la ao coordenador.

Como observado, os vários comandos existentes contêm parâmetros. O parâmetro *Loop, é* um valor booleano que indica se a medição vai decorrer durante um tempo indefinido ou instantâneo. Se a medição está definida para decorrer

durante um tempo indefinido, o administrador pode escolher entre: *Time Feedback*, onde o administrador define o tempo entre as respostas dos *peers* ao coordenador, podendo estes tempos serem alterados pelo parâmetro *Feedback*; e *Statistical Feedback*, que permite ao administrador escolher um conjunto de opções de estatísticas para serem aplicadas quando o administrador cancela um comando de medição a ser feito *probing* pelos *peers*. Outro parâmetro existente é o ICMP, o seu valor de *flag* por defeito é ativo. Com esta *flag* ativa-se o protocolo ICMP para se usar num comando específico. Se a *flag* for desativada pelo administrador, é usada uma implementação feita em Java para executar o comando sem recorrer ao ICMP. O parâmetro que permite criar condições de alarme é o *Alarm Condition* se a *flag Alarm* estiver ativa, o administrador precisa de escolher uma condição lógica (maior, menor, igual a valor) para despoletar o alarme. Todas as entidades foram implementadas em *Java* e para criar a interface gráfica do coordenador foi utilizada a biblioteca *Graphical User Interface* (GUI) *JavaFX*.

III. MECANISMOS DESENVOLVIDOS

Nesta secção serão descritos os passos para a criação de uma metalinguagem para ajudar um administrador a pré-programar a rede de monitorização, como também todos os casos considerados para a otimização do sistema P2P. Por fim, será exposto como o sistema é tolerante a determinadas falhas.

A. Metalinguagem para programação da overlay

Foi criada uma metalinguagem com o objetivo de aumentar a versatilidade e rapidez com que os administradores interagem com o sistema. Esta nova forma de comunicar com o sistema oferece a possibilidade de pré-programar a rede de monitorização através de instruções que podem ser construídas de várias formas, possibilitando a adaptação a diferentes cenários e tornar determinadas ações do sistema autónomas. É possível programar ações como iniciar ou cancelar *probings* numa data especifica, ou quando acontece algo na rede, tornar o sistema reativo. Esta metalinguagem tem o papel de fornecer novas funcionalidades ao sistema, e.g.: *i)* pré-programar a rede com instruções enviadas aos *peers* num tempo definido previamente, permitindo iniciar ou cancelar *probings* numa data especifica ou entre datas; *ii)* condicionalmente iniciar *probings* mediante o número de vezes que a condição de alarme de outros *probings* já existentes foi despoletada, permitindo a adaptação a diferentes cenários; *iii)* inserir uma lista de instruções através de um ficheiro ou numa área de texto que o sistema pode interpretar e executar, rapidamente configurando o sistema. Existem três tipos de instruções na metalinguagem definida, inicio de *probing*, cancelar *probing* e condicional, onde em cada tipo será possível fazer várias combinações e usar diferentes funcionalidades opcionais:

- No tipo de instrução início de *probing*, tem que ser indicado o endereço IP de origem e de destino, a métrica que irá ser indicada na condição de alarme, se vai utilizar ICMP e o tipo de *loop*. Existem também campos opcionais, o número de pacotes a serem enviados, o tamanho desses pacotes, data de início e de fim, ou só data de início continuando o *probing* a correr. Na Figura 2 é apresentado um diagrama que descreve este tipo de instrução;
- Na instrução de cancelar um *probing* é necessário indicar o id do probing, existindo a opção de cancelar numa data especifica;
- Também existe a instrução condicional que, dando um id de um *probing* já a decorrer, caso esse *probing* despolete o alarme um determinado número de vezes, é iniciado uma ou várias novas instruções do sistema.

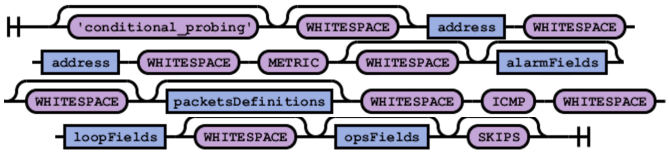

Figura 2. Diagrama dos campos para iniciar um probing

Uma das ferramentas utilizadas na criação da metalinguagem foi o gerador de *parsers* ANTLR, uma ferramenta que ajuda na criação de *parsers* onde se pode ler, processar, executar ou traduzir texto estruturado ou ficheiros binários [5]. No sistema, o input do administrador é convertido numa *Abstract Syntax Tree* (AST), sendo a representação lógica do *input*. Para obter esta AST é necessário definir um *lexer* e uma gramática de *parser*, o *ANLTR* gera ambos, permitindo a sua utilização em diversas linguagens, que neste caso, será em *Java*. Para definir um *lexer* e uma gramática *parser* foi usada a forma de *Backus-Naur Form* (BNF).

B. Otimização do Sistema de Monitorização

Para otimizar o sistema de monitorização, foi necessário obter os valores de várias métricas (e.g. *jitter*, RTT, *packet loss*) de cada link da *path* de cada *probing*, para os valores destas métricas poderem ser reaproveitados em novos *probings* que sejam criados pelo administrador. Por exemplo, se for iniciado um novo *probing* com links que já estão a ser monitorizados por outros *probings* existentes, o coordenador fica encarregue de reaproveitar os valores dos *probings* já existentes para o novo *probing*, sendo aplicadas as condições de alarme se tiverem sido definidas e sem ser necessário enviar comandos de inicio de *probings* a *peers*. A versão inicial do sistema utilizava a chamada de sistema *ping*, que não fornecia o valor das métricas referidas de cada link da *path* de um *probing*, sendo necessário, para obter todos esses valores, iniciar um *ping* por cada link. Para facilitar a obtenção desses valores, foi trocada a chamada de sistema *ping* pela ferramenta *My_Traceroute* (MTR), que fornece a funcionalidade das chamadas de sistema *ping* e *traceroute* [6]. Juntando a informação de ambas, a ferramenta MTR retorna a rota inteira e respetivas métricas observadas em cada link que a compõe, entre dois determinados *peers* da rede de monitorização. Sabendo os valores das métricas referidas de cada link, é então possível agregar os resultados de cada *probing*, não sendo necessário criar mais *probings* do que são estritamente precisos. Previamente era criado um novo *probing* sempre que era necessário saber qualquer umas das métricas. Depois da otimização, como o MTR retorna os valores de todas as métricas necessárias, é só preciso fazer um

probing para retirar todos os valores precisos, quando anteriormente eram iniciados vários *probings*. Para otimizar os *probings* de RTT e de *packet loss*, foram estudados e implementados três casos diferentes onde são reaproveitados valores de *probings* já existentes. Como ilustrado na Figura 3, no *Caso 1*, um *probing* inicial representado a azul, está a ser efetuado entre o *peer R1* até *R5*, de seguida é efetuado um outro *probing*, representado a vermelho, entre o router *R2* e *R4*.

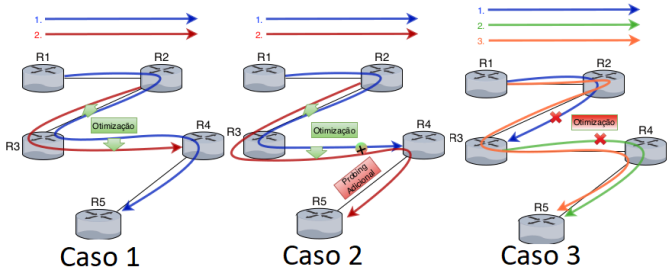

Figura 3. Vários casos da otimização do sistema

Neste caso, não é necessário indicar a nenhum *peer* que é necessário iniciar outro *probing*, é o coordenador que irá iniciar verificações nos dados que o *probing* inicial lhe está a reportar, pois já tem informações de *R2* a *R4* e serão então aplicados os alarmes definidos pelo utilizador, otimizando a rede. Noutro caso, serão também reaproveitados os valores de um *probing* existente, mas será necessário enviar a um *peer* um novo início de *probing* pois o *probing* já existente não contém todos os dados necessários para efetuar o *probing* pedido pelo administrador do sistema. Como ilustrado na Figura 3, no *Caso 2*, é efetuado um *probing* do *peer R1* a *R4*, representado a azul e de seguida é efetuado um *probing* do *peer R2* ao *peer R5*, representado a vermelho. Como o *probing* azul só contém informação do *peer R1* a *R4*, o coordenador terá que iniciar um *probing* adicional para recolher a informação que falta do novo *probing* (*R2* a *R5*), iniciando automaticamente um novo *probing* do *peer R4* a *R5*. Caso o administrador tenha definido uma condição de alarme no *probing*, terá que ser feita a sua verificação no valor final resultante da otimização. No último caso, ao contrário dos anteriores, em vez de serem reaproveitados os valores de *probings* já existentes, estes *probings* serão cancelados e é iniciado um com a *path* maior, que recolhe informações dos *probings* cancelados e outros caminhos. Como ilustrado na Figura 3, no *Caso 3*, existe um primeiro *probing* representado a azul, que efetua medições do *peer R1* ao *peer R3*, existe também outro *probing*, representado a verde, que efetua medições do *peer R3* ao *peer R5*. O administrador posteriormente cria outro *probing*, representado a laranja, que tem início no *peer R1* até ao *peer R5*. Como este *probing* tem uma rota com a *path* maior e inclui o primeiro e segundo *probing*, estes já não serão necessários, sendo cancelados e passando a existir só o último *probing* efetuado. Desta forma, será necessário transportar as definições que o administrador tinha colocado previamente nos *probings* cancelados para os resultados do novo *probing*, para continuarem a funcionar como previamente e a reportar os devidos alarmes, otimizando a rede. Os três casos apresentados podem existir em conjunto, dependendo da combinação de *probings* que um administrador utilize para monitorizar a rede. Para otimizar os *probings* de RTT, é feito o somatório do valor do RTT de todos os links contidos na *path* de um *probing*, podendo ser reaproveitados os valores de *probings* já existentes no coordenador. Em relação à otimização dos valores de *packet loss* é necessário aplicar a Equação (1). Onde a variável l_i é o valor de *packet loss* de cada interface, sendo feita a permutação até ao número total de interfaces resultantes da chamada de sistema do MTR.

$$1-\left(\prod_{i=1}^{interfaces}(i-l_i)\right)=1-\left((1-l_1).(1-l_2)\dots(1-l_{númeroDeInterfaces}-1).(1-l_{númeroDeInterfaces})\right) \quad (1)$$

Foi necessário ter algum cuidado para interpretar os dados resultantes do MTR. Como ilustrado na Figura 4, é possível verificar que na segunda linha, na interface 10.0.27.2 existe *packet loss* e RTT significante. Para poder retirar o valor do RTT da interface 10.0.19.2 é pois necessário retirar o valor do link anterior, neste caso, 82.4 - 21, resultando em 61.4 ms.

Start: 2019-09-25T21:06:42+0100

HOST: n1	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1.|-- 10.0.4.2	0.0%	10	0.2	0.1	0.0	0.2	0.0
2.|-- 10.0.27.2	20.0%	10	21.0	23.0	21.0	33.1	4.1
3.|-- 10.0.19.2	30.0%	10	82.4	82.1	81.6	82.4	0.3
4.|-- 10.0.21.2	50.0%	10	82.8	82.7	81.9	83.6	0.6

Figura 4. Exemplo de um resultado da ferramenta MTR

Para a métrica *packet loss* o raciocínio será diferente, para obter a percentagem de *packet loss* da interface 10.0.19.2 é necessário aplicar a Equação (2), onde a *vInterface* será o valor do *packet loss* da interface 10.0.19.2 (0,3) e a *vInterfaceAnterior* será o valor observado na interface 10.0.27.2 (0,2). Utilizando a Equação (2), o *packet loss* da interface 10.0.19.2 será então 0.125 * 100 = 12,5%.

$$1-\frac{(1-vInterface)}{(1-vInterfaceAnterior)} \quad (2)$$

C. *Tolerância a Falhas do Sistema*

Na versão inicial do sistema, a rede *overlay* não se ajustava caso houvesse, por exemplo, uma mudança de rota devido a alguma falha na rede. De igual forma, caso existissem falhas no coordenador, o sistema inteiro teria que ser restabelecido cancelando todos os *probings* a serem efetuados no momento da falha e refazendo-os posteriormente. Para resolver estes dois casos anteriores foram feitas melhorias ao sistema para o tornar tolerante a falhas que possam ocorrer na rede *overlay*. Para permitir reajustes na interface para correta interpretação dos valores, o coordenador necessita de verificar constantemente se houve alguma mudança de rota nos *probings* que estão a ser efetuados na rede *overlay* e, se houver, efectuar as mudanças necessárias automaticamente, e informar o sistema da troca de rotas para posteriormente o administrador ter conhecimento. Se ocorrer uma falha no coordenador, é necessário alertar os *peers* que o seu estado foi restabelecido para voltarem a enviar os dados dos *probings* que estavam a fazer. O coordenador armazena todos os *probings* que estão ativos na rede num ficheiro para

no caso de falha poder utilizá-lo para repor o seu estado anterior. Tal como o coordenador, os *peers* também terão de ter um ficheiro onde são armazenados todos os *probings* que cada *peer* tem ativos. É pois necessário alertar o coordenador quando um *peer* volta a restabelecer o seu estado.

IV. Análise de Resultados Ilustrativos

Para comprovar o correto funcionamento do sistema, será apresentado um caso de teste demonstrativo do funcionamento dos *probings*. Em relação à metalinguagem criada, será verificado se o sistema executa todas as instruções corretamente. Para a otimização do sistema, serão feitas comparações antes e depois do sistema ser otimizado. Por fim, são apresentados testes para verificar se o sistema é tolerante a falhas. Os testes serão realizados utilizando o emulador de redes *Common Open Research Emulator* (CORE) [8], que permite criar redes e correr aplicações reais nos *hosts* da rede. Foi então emulada uma rede ISP e executados os programas desenvolvidos nos vários peers e no coordenador. O lado esquerdo da Figura 5 ilustra a rede criada no emulador CORE, o lado direito da Figura 5 ilustra a interface gráfica que o coordenador da rede disponibiliza ao administrador do ISP.

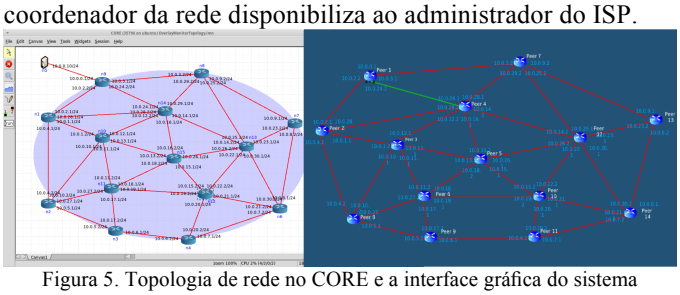

Figura 5. Topologia de rede no CORE e a interface gráfica do sistema

A. Teste de Probing Feedback Loop

Neste caso de teste é apresentado um caso de um *loop probe* com feedback constante (tempo de *feedback* de 5 segundos) e com o tipo de medição *route path*. Este *probing* foi configurado no link entre o *peer3* e o *peer4* e com o alarme configurado para ser acionado quando uma mudança de *route path* ocorrer. Para forçar este cenário, no emulador CORE, o link entre os peers vai ser desligado, forçando assim uma nova rota entre os peers. A Figura 6 mostra claramente que a mudança de *route path* foi detetada pelos alarmes acionados, mostrando a alteração de caminho motivada pelo link que foi desligado na topologia de rede.

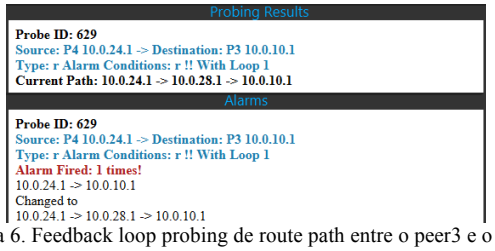

Figura 6. Feedback loop probing de route path entre o peer3 e o peer4

B. Teste de uma Instrução Condicional

Como descrito anteriormente, é possível criar instruções condicionais com a metalinguagem definida. Para este caso de teste, foi criada a instrução *conditional "619 5 10.0.0.1 10.0.4.2 rtt > 100 yes loop 20"*, que irá iniciar um *probing* de RTT de origem o IP 10.0.0.1 e destino 10.0.4.2 (entre outros parâmetros) quando o *probe* com o ID 619 disparar o alarme 5 ou mais vezes, como ilustrado na Figura 7. Caso o administrador se engane no processo de introduzir a instrução, um *popup* é apresentado indicando o tipo de erro, a linha, posição e também o tipo de valor em falta.

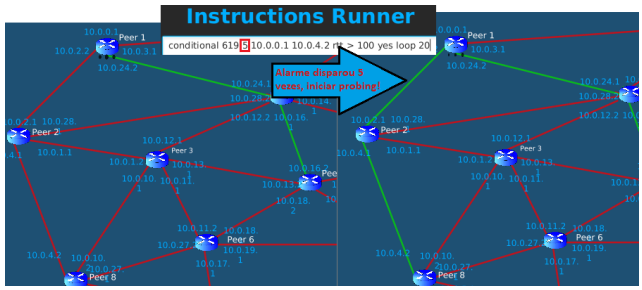

Figura 7. Transformação do sistema ao ser executada uma instrução condicional e a condição de alarme ser alcançada

C. Testes da Otimização do Sistema

Foram feitos testes comparado o sistema sem e com as várias otimizações aos probings. A ferramenta de medição de tráfego *vnStat* foi utilizada para este teste, dando informações sobre cada interface, tal como o envio e receção de bytes, máximo, mínimo e média de envio por segundo, pacotes transmitidos, etc. [7]. O *vnStat* não faz *sniffing* de nenhum tráfego e assegura uma utilização leve, sendo um perfeito candidato para utilização no CORE. Foram executados 12 *probings* utilizando a metalinguagem, uma vez com a otimização outra sem a otimização, durante um período de 10 minutos. Cada probing será de RTT ou *packet loss*, focando nestas duas métricas em particular pois foram as que foram mais otimizadas, sendo possível o reaproveitamento dos valores de outros *probes*. Para gerar algum tráfego na rede os pacotes foram enviados com tamanho de 908 bytes, enviando 20 pacotes a cada 20 segundos, para estar continuadamente a gerar tráfego na rede. Foi executada a ferramenta *vnStat* na interface do coordenador e, posteriormente, nas três interfaces do *peer1* (*n9*). A Figura 8 compara os bytes transferidos no coordenador e no *peer1* com e sem otimização do sistema.

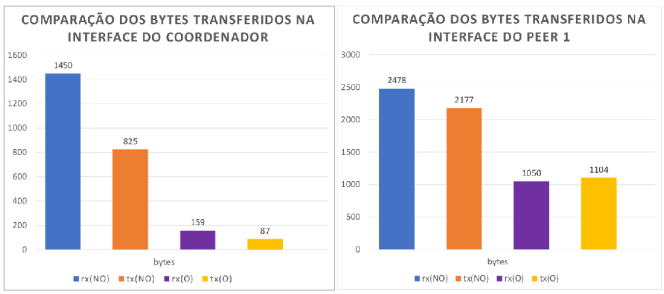

Figura 8. Gráficos comparativos dos bytes transferidos no peer e coordenador

Observa-se uma diminuição de 89% de bytes recebidos e de 89% de bytes enviados na interface do coordenador. E observa-se uma diminuição de 58% de bytes recebidos e de 49% de bytes enviados nas interfaces do *peer1*. Como se pode verificar, a otimização foi bem-sucedida, reduzindo bastante o

tráfego da rede e obtendo os mesmos resultados finais de cada *probing*. A discrepância dos valores deve-se ao facto de na versão inicial do sistema ser enviado o resultado ao coordenador sempre que um *ping* era feito, resultando em tráfego desnecessário a atravessar a rede. Na versão otimizada do sistema, com a ferramenta MTR, só é enviado o resultado de cada *probing* depois de enviar os pacotes definidos pelo administrador entre o *peer* origem e *peer* destino. Previamente, sempre que era criado um *probing*, não eram reutilizados nenhuns valores de outros *probings* já existentes, na versão otimizada só é necessário correr um *probing* para obter resultados sobre as métricas todas de uma *path*, onde anteriormente para obter os resultados de todas as métricas dessa mesma *path* seria necessário correr vários *probings* diferentes. Para casos de *probings* de RTT e *packet loss*, no sistema otimizado, são também reutilizados todos os valores dos links que já estão a ser monitorizados, sendo aplicado um dos três casos de composição de métricas referidos nas secções anteriores,. Assim, *probings* de RTT e *packet loss*, são otimizados mesmo se não existirem *probings* com exatamente a mesma path, basta existirem os valores dos links que compõe a *path* dos novos *probings*; O *payload* da versão inicial do sistema era maior quando ocoria um alarme pois este ia no resultado dos *probings*. Na versão otimizada os *peers* já não precisam de enviar o alarme nos resultados dos *probings* pois é o coordenador que verifica os alarmes ao receber os valores transmitidos.

D. Testes da Tolerância a Falhas do Sistema

Para testar se o sistema se adaptava corretamente a uma falha num link, foi iniciado um *probing* de RTT, do *Peer1* até ao *Peer12* e posteriormente introduzida uma falha no link que interliga o *Peer4* e o *Peer12*. Como ilustrado na Figura 9, o sistema deteta a falha e automaticamente é atualizada a GUI do sistema para refletir a mudança de *path*.

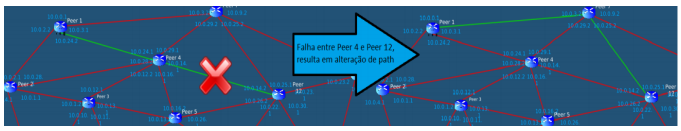

Figura 9. Ilustração da alteração de path na interface do sistema

Para testar se o sistema se tornou tolerante a falhas, foi colocado o sistema em funcionamento com vários *probings* iniciais, sendo posteriormente injetadas falhas no coordenador e em *peers* com o objetivo de verificar se estes retornam ao seu estado anterior quando forem novamente iniciados.

Ao reiniciar o coordenador, desligando abruptamente o processo que corria o sistema e iniciando-o de novo, é apresentado um *popup* indicando algumas informações do seu estado anterior, onde são apresentados quantos *probings* estava a realizar. De seguida, foram reiniciados vários peers de forma a testar se voltavam ao estado anterior, sendo este processo bem-sucedido no seu todo o que comprova a eficiência dos mecanismos implementados.

V. CONCLUSÕES

Este artigo aborda os processos de implementação e demonstração de um sistema P2P de monitorização de redes com vários mecanismos que foram apresentados que o tornam versátil, otimizado, resiliente e com uma interface intuitiva e simples de utilizar por administradores de redes ISP. Apesar de existirem alguns projetos com algumas particularidades relacionadas com este, não existe muita informação relacionada com o desenvolvimento e implementação de sistemas P2P para monitorização de redes, sendo este artigo uma boa contribuição para quem queira fazer algum trabalho relacionado com este tipo de sistemas.

Como possível trabalho futuro, podem ser adicionados mais tipos de *probings*. Também seria interessante estudar a expansão e testar o sistema desenvolvido em cenários envolvendo ambientes multi-ISP, onde vários administradores podem cooperar na gestão das operações das redes e na deteção das anomalias. A interface do sistema pode também ser enriquecida com uma melhor visualização dos alarmes e com novas funcionalidades como adicionar novos *peers* facilmente. Também pode ser feita uma análise dos níveis de alerta gerados e dos tempos de recuperação a falhas da rede mediante os diferentes protocolos de *routing* utilizados pelo ISP (*Routing Information Protocol* (RIP), Open Shortest Path First (OSPF), entre outros).

AGRADECIMENTOS

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Rafiullah Khan and Sarmad Ullah Khan. *Design and implementation of an automated network monitoring and reporting back system*. Journal of Industrial Information Integration, 9:24–34, 2018.

[2] Sihyung Lee, Kyriaki Levanti, and Hyong S Kim. *Network monitoring: Present and future*. Computer Networks, 65:84–98, 2014.

[3] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. *A survey and comparison of peer-to-peer overlay network schemes*. IEEE Communications Surveys & Tutorials, 7(2):72–93, 2005.

[4] Miguel Silva, Ricardo Mendonça, and Pedro Sousa. *A P2P overlay system for network anomaly detection in ISP infrastructures*. In 2019 14th Iberian Conference on Information Systems and Technologies (CISTI), IEEE, 2019.

[5] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013

[6] MTR. http://www.bitwizard.nl/mtr

[7] vnStat. https://humdi.net/vnstat

[8] CORE Emulator. https://www.nrl.navy.mil/itd/ncs/products/core

[9] RIPE Atlas. https://atlas.ripe.net/

[10] perfSONAR. https://www.perfsonar.net/

[11] Sam Knows. https://www.samknows.com/

[12] NLNOG RING. https://ring.nlnog.net/